1 Objectives

There are many choices available to the language designer when a function call is made. The choices made will have a significant effect on the language.

Your objectives for this lecture:

- Show how to implement thunks by using local state and user-defined types.
- Show how thunks can implement the call-by-need parameter-passing style.
- Show how to create virtually-infinite data-structures by using lazy evaluation.

2 Announcements

3 Examples

```
# type 'a status = Value of 'a
                 | Susp of (unit -> 'a);;
# let delay f =
    let status = ref (Susp f) in
    fun () -> match (!status) with
    | Value a -> a
    | Susp f -> let result = f () in
                ( status := (Value result);
                  result );;
val delay : (unit -> 'a) -> unit -> 'a = <fun>
# let force f = f ();;
```

```
type 'a llist = Cons of 'a * 'a llist Lazy.t | Nil
# let rec ftake n llist =
  match n,llist with
  | _,Nil -> []
  | 0,_ -> []
  | _,(Cons (x,xs)) -> x :: ftake (n-1) (force xs)
val ftake : int -> 'a llist -> 'a list = <fun>
# let rec ones = Cons(1, lazy ones);;
val ones : int llist = Cons (1, contents = Delayed <fun>)
# let rec numsfrom n = Cons(n, lazy (numsfrom (n+1)));;
val numsfrom : int -> int llist = <fun>
```

Problems on the back...

4 Problems

Try the following problems. In a few minutes the instructor will go over the solutions. Feel free to work with the person next to you!

1. Write another version of `nats` (the list of natural numbers) without using a function.
2. Write a function `circular` that takes a normal list and returns an infinite circular list with the same data. (This one is a bit tricky.)

```
# let ott = circular [1;2;3];;
val ott : int llist = Cons (1, contents = Delayed <fun>)
# ftake 10 ott;;
- : int list = [1; 2; 3; 1; 2; 3; 1; 2; 3; 1]
```

3. What happens if we pass `[]` to `circular`?